Lab Exercises: Introduction to Object-Oriented Programming

1. Implement a class called Sphere to represent a solid geometric sphere. Your class should have the following methods:

   `__init__(self, radius)` Creates a sphere having the given radius
   `getRadius(self)` Returns the radius of this sphere
   `surfaceArea(self)` Returns the surface area of the sphere
   `volume(self)` Returns the volume of the sphere

   The following formulas will help:
   $\textit{Surface Area} = 4 \pi r^2$
   $\textit{Volume} = 4/3 \pi r^3$

   Test your class by creating a new sphere of radius 1 and asking it for its surface area and volume:
   ```
   >>> s = Sphere(1)
   >>> print s.surfaceArea()
   12.566370614359172
   >>> print s.volume()
   4.1887902047863905
   >>>
   ```

2. Implement a class called Car with the following properties. A Car has a make, model, and year, a certain fuel efficiency (measured in miles per gallon), and a certain amount of fuel in the gas tank (measured in gallons). First, we need a constructor method `__init__` that takes the make, model, year, and miles-per-gallon and initializes the Car's internal variables accordingly:

   ```
   class Car:

      def __init__(self, make, model, year, mpg):
         self.carMake = make
         self.carModel = model
         self.carYear = year
         self.efficiency = mpg
         self.gas = 0.0
   ```

   We can then create a new car object like this: `mycar = Car(“Honda”, “Accord”, 2004, 28)`

3. Next, let's add a `__str__` method so that we can print out Car objects in a nice way. We'll have `__str__` return a string such as “2004 Honda Accord”:

   ```
     def __str__(self):
         s = “%d %s %s” % (self.carYear, self.carMake, self.carModel)
         return s
   ```

   Now test your code by creating a few Car objects and printing them:

   ```
   mycar = Car(“Honda”, “Accord”, 2004, 28)
   myothercar = Car(“Rolls Royce”, “Phantom”, 1968, 15)
   print mycar
   print myothercar
   ```

(continued on back)

4. Next, add methods called `refuel` and `drive` to your Car class. The refuel method should take an amount of gas (in gallons) as input and add it to the car's internal fuel level. The drive method should take a distance (in miles) as input and reduce the car's gas by the amount needed to drive the given distance. Hint: the amount of gas needed to drive *d* miles is *d* / *mpg*, and the distance traveled on *g* gallons of gas is *g* × *mpg*. If the fuel needed is greater than the amount available in the tank, it should be set to zero and a message "Ran out of gas" should be printed. Your methods should behave as shown below:

```
>>> mycar = Car("Honda", "Accord", 2004, 28)
>>> mycar.refuel(10)
Added 10 gallons of gas to the tank
>>> mycar.gas
10.0
>>> mycar.drive(150)
Drove 150 miles.  4.6 gallons of gas left
>>> mycar.drive(200)
Ran out of gas after 130 miles.
>>> mycar.gas
0.0
```

5. Implement a class called Card to represent a playing card. Your class should have the following methods:

   `__init__(self, rank, suit)` Creates a card of the specified rank and suit, where `rank` is an integer in the range 1-13 indicating the ranks Ace through King, and `suit` is the string "Spades", "Diamonds", "Hearts", or "Clubs".
   `getRank(self)` Returns the rank of the card
   `getSuit(self)` Returns the suit of the card
   `BJvalue(self)` Returns the Blackjack value of a card. Ace counts as 1, face cards count as 10.
   `__str__(self)` Returns a string that names the card. For example: "Ace of Spaces" or "2 of Clubs".
   `equals(self, otherCard)` Returns true if this card has the same rank and suit as `otherCard`.

   Test your Card class with a program called `testcards()` that prints out *N* randomly generated cards and their associated Blackjack value, where *N* is a number supplied by the user.

6. Implement a Student class. A Student has a name and a total quiz score, which is the sum of all of the quiz scores received by the student so far. Implement the following methods:

   `__init__(self, name)` Creates a student with the specified name with an initial quiz total of 0.
   `name(self)` Returns the student's name
   `recordQuiz(self, score)` Adds a new quiz score to the student's total
   `getTotalPoints(self)` Returns the total points received so far
   `getAverageScore(self)` Returns the student's average quiz score. For this one, the Student object also needs to keep track of the number of quizzes that the student has taken so far.

   Test your Student class as follows:

```
>>> sue = Student("Sue")
>>> sue.recordQuiz(10)
>>> sue.recordQuiz(15)
>>> sue.recordQuiz(14)
>>> print sue.getTotalPoints(), sue.getAverageScore()
39 13.0
```